Optimizing Change Detection in Distributed Digital Collections

An Architectural Perspective of Change Detection

Lakmal Meegahapola, Roshan Alwis,
Eranga Nimalarathna, Vijini Mallawaarachchi,
Dulani Meedeniya
Department of Computer Science & Engineering,
University of Moratuwa, Sri Lanka
{lakmalbuddikalucky.13, alwisroshan.13, eranga.13,
vijini.13, dulanim}@cse.mrt.ac.lk

Sampath Jayarathna
Department of Computer Science,
California State Polytechnic University
Pomona, CA 91768
ukjayarathna@cpp.edu

***Abstract*— Digital documents are likely to have problems associated with the persistence of links, especially when dealing with references to external resources. People keep track of various webpages of their interest using distributed digital collections and without possession of these documents; the curator cannot control how they change. In the current context, managing these distributed digital collections and getting notifications about various changes have become a significant challenge. In this paper, we address the architectural aspects of change detection systems and present optimized change detection architecture, including a web service and a browser plugin, along with an email notification service. We have performed an experimental study on our hybrid architecture for change detection in a distributed digital collection. The proposed method introduces a preliminary framework that can serve as a useful tool to mitigate the impact of unexpected change in documents stored in decentralized collections in the future.**

Keywords— Change detection; change notification; distributed digital collections;

I. INTRODUCTION

Change detection and notification (CDN) refers to the automated process of detecting changes made to webpages and notifying interested users [1]. Prior to the introduction of this concept, users had to manually check for changes by refreshing frequently. With the introduction of CDN systems, users could monitor webpages and easily detect changes occurred. Google Alerts [2] and Follow That Page [3] are two of the most popular change detection services being used at present. Google Alerts is a popular CDN service which monitors the Web for new content. Follow That Page is another widely used CDN service where users can keep track of changes occurred in their favorite webpages. However, many performance issues have arisen within current change detection systems due to the increasing number of websites being tracked and vast number of users being registered every day [4] [5] [6].

Change detection in web pages can be classified into two parts based on the architecture of the detection system. Namely they are Server side detection and Client side detection. In the change detection architecture where server side detection is used, the processing load for the server increases when there is a vast number of web pages. This is because the server has to crawl each and every web page and detect any changes that have been occurred. Additionally, the detection frequency for a web page may decrease due to the large number of new web pages which the server has to track. Scaling of server based change detection solutions has become tedious and expensive as it requires more computational power and resources.

The client side architectural approach uses client machines to poll the web pages and track their changes. When considering existing change detection systems which run with client side detection, the users with the spare computing resources may detect frequent changes to web pages whereas, the rest of the users might not detect these changes. There is a challenge on how to improve client side detection without increasing the load to the client machines because these kind of detection processes run in background. Therefore, with the availability of web browser plugins, client side detection can be improved. Further, the system will consist of a large number of clients crawling web pages across distributed networks every day and they have to be managed efficiently. Hence load balancing mechanisms need to be implemented as well [7].

Each of these different architectures has their own merits and limitations when considering factors such as performance, computational power and speed of detection. There is a need for a solution which optimizes change detection and combines the advantages of client-based and server-based change detection systems while eliminating their weaknesses. Our current efforts continue to study the limitations of the existing change detection systems and propose an efficient solution.

This paper presents a solution with an optimized change detection system via novel hybrid architecture. Section II outlines the related work that has been carried out in the area of interest. Section III describes the proposed system architecture and Section IV explains the change detection and notification methodology and the experiments we carried out. Section V presents the experimental results. Finally, Section VI concludes the paper with possible future research directions.

978-1-5090-5504-3/17/$31.00 ©2017 IEEE
SNPD 2017, June 26-28, 2017, Kanazawa, Japan

II. BACKGROUND

Among many related literature, [8] has discussed how information retrieval has become crucial in web services. Furthermore, Nadaraj [5], has described an approach for distributed content aggregation and change detection for web content using client resources. In that approach, the consumers get the data from a queue and run the crawler on the working machine. It distributes the work among consumers and the results will be aggregated, improving the efficiency of the crawling mechanism. It also reduces the coupling of spiders to machines, allowing them to operate in a distributed network and consider as a scalable content classification approach. Bloom filters have been used to find the duplicate URLs and content in the site. However, bloom filters only confirm that the URL was not visited before.

An interesting work has presented in [6], that focuses on classifying webpages in a digital collection and detecting the changes occurred to them. This method records the differences between two versions of a resource in a digital collection to detect the changes in research conference web sites. A layered architecture for a mobile web based application is proposed in [9]. This has described the importance of reducing client side resource utilization. Another approach for real time scheduling based on multi-agent technologies and multithreading application is described in [10]. This approach allows to obtain efficient and fast solutions to complex problems in real-time.

Prieto et al. [11], present a web change detection system for distributed and collaborative web sites. Mainly search engines use this method to decide when to crawl web pages and build their indexes. Another approach for parallel crawling of the web sites using multiple machines is presented in [12]. They have provided a three-step algorithm to detect changes in a web page. However, the clients are not users of the system but are server nodes. Hence to scale the system, the number of server nodes need to be increased, which will result in high costs.

ChangeDetection.com [13] and 'Follow That Page' [3] are free web services that provide change detection and monitoring facilities for Internet users. Users can register to these services by entering the page URL and email address to which notifications are to be sent. However, if the user has a large number of webpages to track, certain performance issues may occur and users will not get notified frequently.

Distill Web Monitor [14] (formerly known as AlertBox) is a Firefox plugin, which monitors web pages for changes and sends notifications by email and SMS. However, with this plugin, certain users may not detect the changes as frequently as the rest of the users. Another popular desktop application for change detection is Copernic Tracker [15] which keeps track of web pages. This application facilitates to keep track of any number of web pages. This allows keeping track of changes done to words, links or images. There are four pre-set tracking schedules available in this application, namely, multiple times per day, on a daily basis, on a weekly basis and on a monthly basis. Here, alerts can be customized based on tray icons, desktop alerts, SMS notifications and email reports.

Website Watcher [16] is a commercial desktop application which can track unlimited number of pages. It also provides the option to ignore HTML tags, images/banners, numbers and dates. Pages can be checked once a day, once a week, or on a specified day or days of the week. It also allows specifying the checking frequency during a day either in hours or minutes. This is an advantage as the user can adjust the checking frequency. However, allowing the user to adjust the checking frequency could be a limitation, as some changes may be missed due to the wrong judgment by the user.

In this work, we have conducted an initial survey [17] to identify the participants who are interested in detecting webpage changes. Following are the survey questions:

1. Have you ever had the need to track a webpage for a change?
2. Have you ever kept refreshing a page for a long time, expecting a change to occur?
3. Have you ever used any webpage change detection tool like "Google Alerts" or "Follow that Page"?
4. Would you like to have an efficient tool to track changes in a webpage?

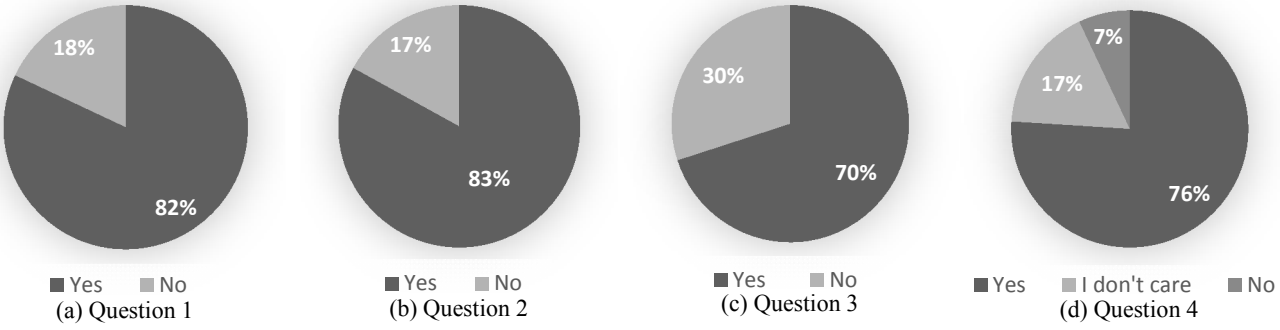

Fig. 1. Answer analysisof the survey questions.

The sample contains 200 randomly selected users. The results [18] show that out of the sample, 82% had the need to track changes in web sites as given in Fig. 1 (a), 83% of them kept refreshing some page expecting a change to occur, as shown in Fig. 1 (b). Only 70% of the participants were actually used a tool to track the changes, which is depicted in Fig. 1 (c). According to Fig. 1 (d), 76% of the sample liked to have an efficient tool to track changes in web pages.

III. SYSTEM ARCHITECTURE

A. Overview

The efficiency of change detection of web pages and related algorithms can be increased using methods such as tree comparison techniques [19] and diff algorithms [20] . It is clear that currently researchers have reached a level where further improvements done to algorithms are not quite sufficient to create a significant impact. Hence in our methodology, we shift our focus from the traditional approach of improving algorithms to the architectural aspects of change detection systems. We have implemented a hybrid solution, combining server side and client side change detection using available change detection algorithms. With this hybrid architecture, the change detection mechanism could be made efficient as most of the changes in the web can be detected almost immediately.

B. Hybrid Architecture

Fig. 2, shows our hybrid solution considered for this research. It comprises of 3 main components.

- Web application with a web service.
- Web browser plugin for client side change detection.
- Email notification service.

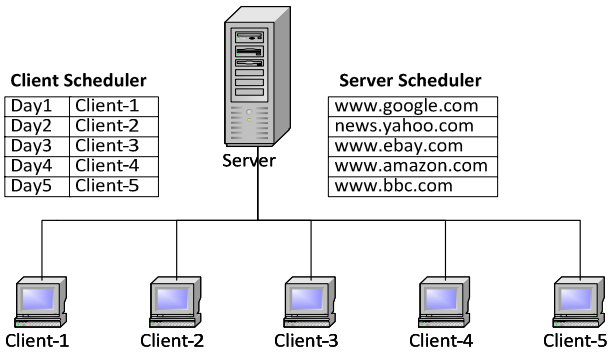

Fig. 2. High level view of the hybrid architecture.

According to the hybrid architecture server has a list of web pages that it has to crawl. When a client comes to our web application and include a new web page for the server to track, it initially run an algorithm to identify the frequency [21] at which changes occur in that particular web page. Once this is done, the web page is added to the list of web pages the server has to poll. But when this list grows further, the server cannot handle the load. It either has to scale up which of course has the issues mentioned in the previous sections. Hence the client side detection comes into help.

The server needs to prepare a new schedule for the clients who have already requested to monitor this particular page by including the new client in the schedule. Server prepares the new schedule and let the clients know about the new polling times and intervals. Fig. 2, presents this stage where server has a schedule and the clients who have subscribed for a particular web page have their own schedule. During scheduling, the server detect changes in web sites in the list using the Machine Learning based algorithm [6], where it directly tries to identify both relevant, irrelevant changes and type of changes as well. If a relevant change is detected, all the clients are notified about the change through an email.

At the same time the clients go on polling the particular web pages they wanted to track according to the predefined schedule. Clients run a light weight algorithm to detect the changes and when a change is detected by a particular client, it directly notifies the server about the change. Then the server run the machine learning based algorithm on that particular web page to identify relevancy of the changes occurred. If relevant changes are more than the threshold, it goes through the routine and notifies all the clients through an email. With this new architecture, the time between two server polls is divided between the clients according to a scheduling algorithm, which makes sure that all the clients would get updates about any new changes almost as soon as they occur.

C. Change Detection

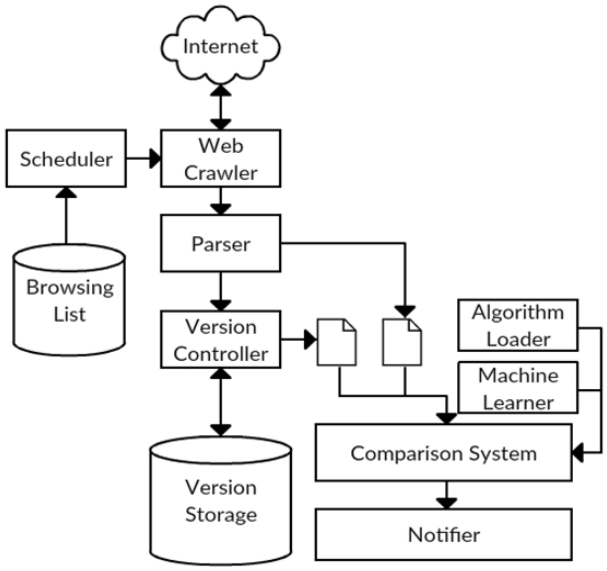

Fig. 3. Server side change detection mechanism.

Fig. 3, presents the work flow of the server side change detection mechanism. It describes how the components including Scheduler, Web Crawler, Parser and Notifier work as a single unit showing relationships between each component as well. Given below is the terminology used in Fig. 3.

- Browsing List: Storage of URLs which users have requested to monitor. A URL maps to set of clients who are interested in particular URL
- Scheduler: Generate a timetable for individual clients to process.
- Web Crawler: Retrieve web contents via internet.
- Version Controller: Manage versions of web archives in the system (handle CRUD operations)

- Version Storage: Collection of versions of web archives.
- Algorithm Loader: Collection of algorithms to detect changes.
- Machine Learner: Machine learning model which can classify the type of the change occurred.
- Comparison System: Compare different versions of same web contents and evaluate the change.
- Notifier: Notify clients who are interested in a particular change.

Initially system has an empty browsing list (Linked List). Server will wait until client requests to occur. If Browsing List is not empty Scheduler will process in regular intervals. Scheduler is responsible for allocating time to run the change detection algorithm among different clients who are interested in the same source. If a client detects a change, client will notify the server and add client request to the user generated linked list.

Apart from the client schedule, server has a separate schedule to process linked list and user generated linked list. In a single iteration server, will process one instance from each list. Using the Web Crawler, server loads the Web content of particular URL to the server environment. Contents will then parse into a HashMap using the Parser and generate the current version. Then old version for particular URL will retrieved from the version storage via version controller. Both versions then submit to the comparison system. If server detects a change exceeding particular threshold, it will use machine learner component to classify the type of change that occurred. Finally, notifier will notify relevant clients about the change.

IV. METHODOLOGY

As shown in Algorithm 1, the process consists of change detection and classification of the websites using the hybrid architecture. There are two main process queues linked list (usual crawling queue by server) and user generated linked list (links of websites detected as 'changed', by clients). In each cycle, server will be operated on each queue for one instance. We then selected 20 frequently changing websites. For a particular web page, we first run a lightweight algorithm (O(n) time complexity) to detect whether there is any change. If there is, then we run a more complex and sophisticated, machine learning based algorithm (O(n2) time complexity) to get more knowledge on the type and relevancy of the change happened.

The light weight algorithm which we used to detect changes generates a hash value for each leaf node in html tree structure of the particular web page. Then it compares the changes of hash values of leaves in older tree version with the newer version. From this comparison, it could obtain the changed nodes, unchanged nodes, new nodes and missing nodes. Note that we do not describe the change detection algorithms that get executed once a change is detected in this research paper, since our research area is on the architectural aspect of change detection. A detailed description of the change detection algorithms employed is beyond the scope of this paper and can be found in [6] [20].

Using two change detection algorithms, we tested the performance of server-side change detection architecture (SSA) and client-side change detection architecture (CSA). Then we compared those performance figures with proposed hybrid architecture (HA).

Algorithm 1 Change Detection and Notification

Require: Dataset of links
Ensure: Changes in Web pages and notify

```
1.  Function mlClassifier(link, new_version, old_version)
2.      set X = extractFeatures(new_version, old_version)
3.      Y = MLClassificationModel(X)
4.      client_list = getClients(link)
5.      setNewVersion(link, new_version)
6.      notify(client_list, Y)
7.  Function isDifference(link, new_version, old_version,)
8.      threshold = getThreshold(link)
9.      diff = difference(new_version, old_version)
10.     return threshold < diff ? True : False
11. while True do
12.     link = getNextFromLinkList()
13.     new_version, old_version = getVersions(link)
14.     if isDifference(link, new_version, old_version) then
15.         mlClassifier(link, new_version, old_version)
16.     end if
17.     link = getNextFromUserGeneratedLinkList()
18.     new_version, old_version = getVersions(link)
19.     mlClassifier(link, new_version, old_version)
20. end while
```

The experiments were conducted on a cluster of 11 Virtual Machines (VMs) which were running in Azure private cloud. Each VM had Linux Ubuntu (kernel 3.13.0-36-generic). 10 of the VMs (acted as clients) were running 64-bit IntelTM Intel Xeon E312xx (Sandy Bridge) which operates at 2.70GHz. They had one CPU socket, with 2 cores each. L1(d/i) and L2 caches were 32KB and 4096KB respectively. Each client had 4GB RAM with one 40GB hard disk. Server VM was running 64-bit IntelTM Intel Xeon E312xx (Sandy Bridge) which operates at 3.4GHz. It had four CPU sockets, with 2 cores each. L1(d/i) and L2 caches were 64KB and 8192KB respectively and RAM was 16GB. In each test, we collected Java Flight Recorder [22] dumps to analyze CPU, memory and network usage.

V. RESULTS AND DISCUSSION

In the client side architecture, since each client has to crawl and detect changes on its own and it requires more CPU and memory than the proposed hybrid architecture (See Table 1). Further from the spike pattern shown in Fig. 4 (a), (b) and (c); in the proposed HA system, when the number of clients who are subscribed for a particular web site doubled from 5 to 10, the detection frequency of a client got halved without affecting the overall change detection frequency. In fact the overall detection frequency got higher when more clients were added because the time between two server polls were divided between more clients to poll that particular website.

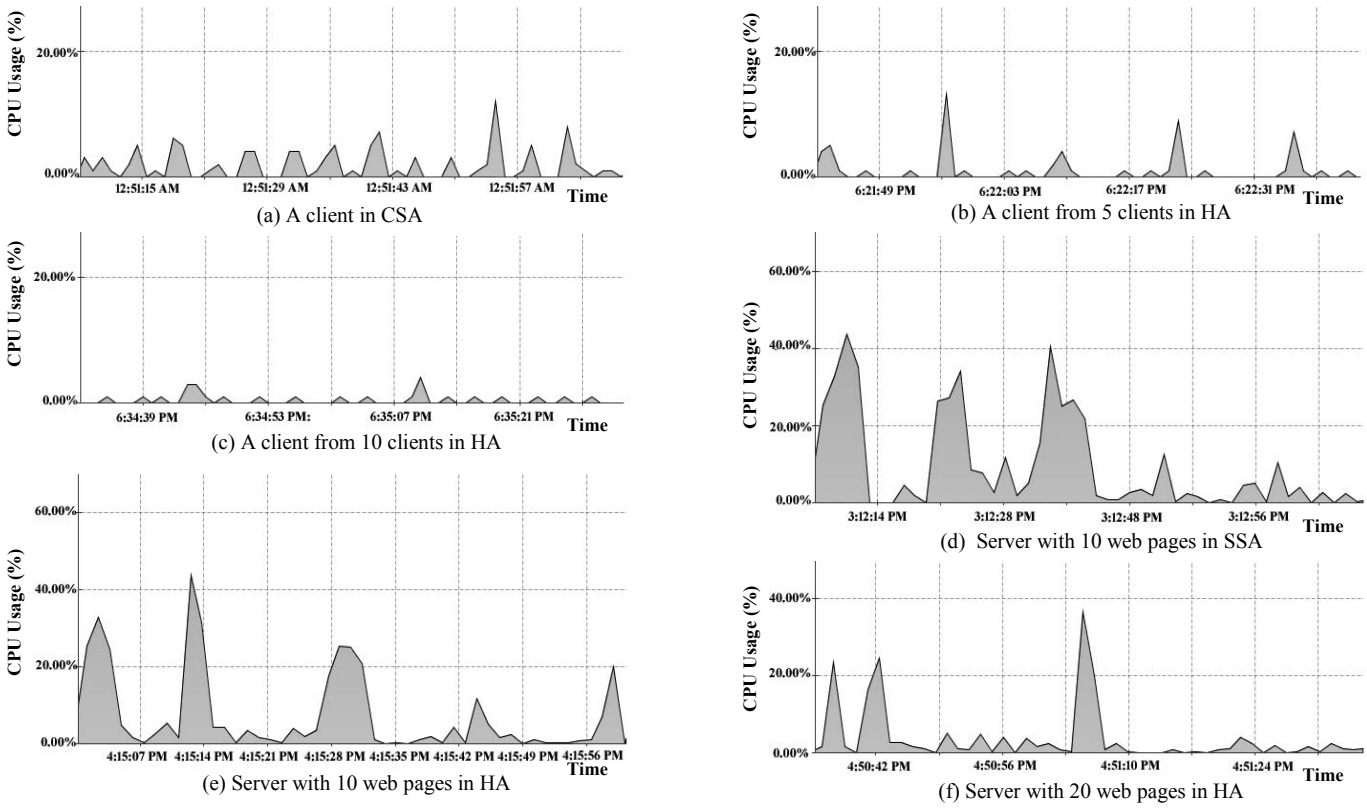

Fig. 4. CPU usage in CSA, SSA and HA Models

According to the results in Table 2 and Fig. 4 (d), (e) and (f); resource utilization is not much different in the server of both SSA and HA even when the number of webpages to track is doubled. The reason for that is regardless of the number of web sites, server needs to crawl through its web page list repeatedly anyway.

TABLE I. AVERAGE RESOURCE UTILAZATION OF A CLIENT IN CLIENT SIDE ARCHITECTURE VS. HYBRID ARCHITECTURE

Average usage	Any Number of Clients (CSA)	5 Clients (HA)	10 Clients (HA)
CPU	3.84%	1.64%	0.966%
Memory	45MiB	39.4Mib	31MiB

Note that the details of a server with 20 web pages in SSA is not shown here because in SSA, regardless of the number of web pages it has, the server need to continue crawling and detecting changes. Hence the spike pattern and resource consumption would be same regardless of the number of web pages it has to track. However, the change detection frequency of a particular web site is still much higher in HA since its clients crawl for changes in parallel to the server. Also note that with the proposed HA, the server doesn't have to scale up in order to make the detection process faster and to reduce the overall time taken to complete one round of the web page list.

TABLE II. AVERAGE RESOURCE UTILIZATION OF THE SERVER IN SERVER SIDE ARCHITECTURE VS. HYBRID ARCHITECTURE

Average usage	10 Web Pages (SSA)	10 Web Pages (HA)	20 Web Pages (HA)
CPU	31.1%	34.2%	32.1%
Memory	234MiB	234MiB	236MiB

In a real-world scenario, this advantage could be crucial, since the number of web pages a server has to track could be more than 100,000. According to the results it is clearly visible that even incase all the clients are not online which means in the worst-case scenario, still the system would behave as good as the server side detection system. But in scaling with many clients getting into the system, it is highly unlikely the worst-case scenario may occur. Hence, we can always expect the system to work in a very high efficiency compared to currently available systems. Even though these results are for a set of 20 web pages and 11 Azure VMs, the solution can be scaled to cater a large number of web pages

and computing devices as scheduling and architecture were developed giving priority to scalability.

Another rare case which causes worst-case would be clients with different web pages to track. This means if there are 10 clients, all of them have 10 different web pages to track and altogether there are 100 web pages without any overlaps. In that case, the server would still consume the same resource limit as a server in SSA crawling and detecting changes. Since there is at least one client for each page the change detection frequency would be better in HA than SSA. In this scenario, the clients in HA would have to crawl and detect changes in same interval as the server. However in HA, clients do not use a complex algorithm like server hence the resource utilization of a client would still be lower than a client in CSA. Hence HA performs well in these kind of worst cases scenarios..

VI. CONCLUSION

Currently there is a clear need for people to keep track of changes occurred in webpages and much efficient solutions are needed. Change Detection and Notification (CDN) systems have made a huge impact on the area of information retrieval by automating the process of change detection. However, still there are opportunities for improvement on the current implementations of CDN systems when considering performance and speed of detection. This paper proposed a hybrid architecture that supports the optimization of current change detection systems. This novel approach can make a significant impact in the field of change detection and notification of web pages.

We have performed an experimental study for a hybrid architecture combining client side architecture and server side architecture for change detection in web content. The experimental results show that there is a significant performance improvement in the process of change detection using the proposed HA compared to traditional CSA and SSA models. This work can be extended by using this hybrid architecture for cases where there are multiple distributed servers instead of the single server which we have considered during this work. Further improvements in the scheduling algorithm can be done in the future in order to enhance the efficiency of the scheduler which is used currently.

REFERENCES

[1] S. Chakravarthy and S. Hara, "Automating Change Detection and Notification of Web Pages (Invited Paper)," in 17th International Workshop on Database and Expert Systems Applications, Krakow, Poland, 2006.

[2] "Google Alerts - Monitor the Web for interesting new content," [Online]. [Accessed 8 February 2017].

[3] "Follow That Page - web monitor: we send you an email when your favorite page has changed," [Online]. Available: https://www.followthatpage.com. [Accessed 8 February 2017].

[4] D. Yadav and A. K. Sharma, "Change Detection in Web Pages," in 10th International Conference on Information Technology, Rourkela, India, 2007.

[5] S. Nadaraj, "Distributed Content Aggregation& Content Change Detection using Bloom Filters," International Journal of Computer Science and Information Technologies,, vol. 7, no. 2, pp. 745-748, 2016.

[6] S. Jayarathna and F. Poursardar, "Change Detection and Classification of Digital Collections," in 2016 IEEE International Conference on Big Data, Washington D.C., USA, 2016.

[7] S. Fujita, "Load Balancing of Peer-to-Peer MMORPG Systems with Hierarchical Area-of-Interest Management," International Journal of Networked and Distributed Computing, vol. 3, no. 3, pp. 177-184, 2015.

[8] M. S. Cha, S. Y. Kim, J. H. Ha, M.-J. Lee, Y.-J. Choi and K.-A. Sohn, "Topic Model based Approach for Improved Indexing in Content based Document Retrieval," International Journal of Networked and Distributed Computing, vol. 4, no. 1, pp. 55-64, 2016.

[9] B. A. Kumar, "Layered Architecture for Mobile Web Based Application: A Case Study of FNU Student Registration System," International Journal of Software Innovation, vol. 4, no. 3, pp. 51 - 64, 2016.

[10] Y. Shepilov, D. Pavlova and D. Kazanskaia, "Multithreading MAS Platform for Real-Time Scheduling," International Journal of Software Innovation, vol. 4, no. 1, pp. 48 - 60, 2016.

[11] V. M. Prieto, M. A. lvarez, V. Carneiro and F. Cacheda, "Distributed and Collaborative Web Change Detection System," Computer Science and Information Systems, vol. 12, no. 1, pp. 91-114, 2015.

[12] D. Yadav, A. Sharma, J. Gupta, N. Garg and A. Mahajan, "Architecture for Parallel Crawling and Algorithm for Change Detection in Web Pages," in 10th International Conference on Information Technology, Orissa, India, 2007.

[13] "ChangeDetection - Know when any web page changes," [Online]. Available: https://www.changedetection.com/. [Accessed 4 Feburary 2017].

[14] "Distill Web Monitor," [Online]. Available: https://addons.mozilla.org/en-us/firefox/addon/alertbox/. [Accessed 4 February 2017].

[15] "Copernic - Tool to track changes on web pages," [Online]. Available: http://www.copernic.com/en/products/tracker/. [Accessed 25 March 2017].

[16] "WebSite-Watcher - Software to check websites for updates and changes (web page monitoring)," [Online]. Available: http://aignes.com/. [Accessed 25 March 2017].

[17] "Survey on Change Detection in Webpages," [Online]. Available: https://vijinim.typeform.com/to/dlnPdY. [Accessed 14 February 2017].

[18] "General report - Survey on Change Detection in Webpages," [Online]. Available: https://vijinim.typeform.com/report/dlnPdY/kcXx. [Accessed 5 March 2017].

[19] S. D. Jain and H. Khandagale, "A Web Page Change Detection System For Selected Zone Using Tree Comparison Technique," International Journal of Computer Applications Technology and Research, vol. 3, no. 4, pp. 254 - 262, 2014.

[20] Y. Wang, D. J. DeWitt and J. Y. Cai, "X-Diff: an effective change detection algorithm for XML documents," in 19th International Conference on Data Engineering, Bangalore, India, 2003.

[21] D. Ford, C. Grimes and E. Tassone, "Keeping a Search Engine Index Fresh: Risk and optimality in estimating refresh rates for web pages," in Proceedings of the 40th Symposium on the Interface: Computing Science and Statistics, Durham, NC, USA, 2008.

[22] "Java Flight Recorder Runtime Guide," [Online]. Available: https://docs.oracle.com/javacomponents/jmc-5-4/jfr-runtime-guide/run.htm#JFRUH164. [Accessed 27 February 2017].